For office use only	Team Control Number	For office use only
T1 _____	**1922476**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**D**	F4 _____

2019
MCM/ICM
Summary Sheet

Summary

A tall tree catches the wind. For such a world famous tourist attraction as the Louvre, it is essential to take precautions against threats ranging from terror attacks to fire. Therefore, our team is asked to establish an emergency evacuation model for Louvre.

We first transform each floor of the Louvre into **an undirected graph** in order to conduct further research from a macro perspective, regarding every group of people as an integral and ignoring individual behaviors.

Next we divide the time cost into two parts: (1)The time spent traveling from each starting point to the stairway or exit on the same floor; (2)The time spent passing the exits or going upstairs / downstairs. For the first part, we employ the **classic Floyd algorithm** to solve the distance between every two nodes in the undirected graph. For the second part, we add a constraint to the flow of people through exits or stairs. Obviously when two groups of people meet up at exits or stairs, one group have to wait until the other have passed.

Then we attempt to determine the best allocation plan by calculating the proportion of people from the same starting point going to each exit or stairway. Since multi-source dynamic network flow problem proves to be NP complete,we use **Improved Annealing Algorithm** to obtain optimal solutions. For the case of multiple floors, we let people wait until the floor nearer to the exits has emptied. The result basically conforms to our intuition that the majority of people will reach the nearest stairway or exit (while we still need some people to go to farther stairways or exits in order to minimize total time cost), and the total time cost for groups on the same floor should be close.

In the meanwhile, we conduct bottleneck analysis for congestion at nodes or stairs respectively. We obtain the variation of number of people at nodes or stairs over time, and we use R and matlab to plot the curves of variation. Then we can tell the potential bottlenecks from the curves by analyzing the places where a great many of people gather for a long time. The result shows that bottlenecks **at nodes** tend to appear near stairs or exits, while the bottlenecks **at stairs or exits** seem to appear randomly.

To test the efficiency of our evacuation model, we compare our strategy with the simple straight-forward strategy, which means everyone rushes to the nearest exit or stairway. The result shows that the time cost of our strategy is 40% less than the simple strategy, which demonstrates our model's efficiency. Besides, we apply our model on another museum and get an ideal result. This implies the universality of our method.

Keywords: Undirected Graph, Floyd Algorithm, Multi-source Dynamic Network Flow, Improved Annealing Algorithm, Bottleneck Analysis

January 28, 2019

Summary

A tall tree catches the wind. For such a world famous tourist attraction as the Louvre, it is essential to take precautions against threats ranging from terror attacks to fire. Therefore, our team is asked to establish an emergency evacuation model for Louvre.

We first transform each floor of the Louvre into **an undirected graph** in order to conduct further research from a macro perspective, regarding every group of people as an integral and ignoring individual behaviors.

Next we divide the time cost into two parts: (1)The time spent traveling from each starting point to the stairway or exit on the same floor; (2)The time spent passing the exits or going upstairs / downstairs. For the first part, we employ the **classic Floyd algorithm** to solve the distance between every two nodes in the undirected graph. For the second part, we add a constraint to the flow of people through exits or stairs. Obviously when two groups of people meet up at exits or stairs, one group have to wait until the other have passed.

Then we attempt to determine the best allocation plan by calculating the proportion of people from the same starting point going to each exit or stairway. Since multi-source dynamic network flow problem proves to be NP complete,we use **Improved Annealing Algorithm** to obtain optimal solutions. For the case of multiple floors, we let people wait until the floor nearer to the exits has emptied. The result basically conforms to our intuition that the majority of people will reach the nearest stairway or exit (while we still need some people to go to farther stairways or exits in order to minimize total time cost), and the total time cost for groups on the same floor should be close.

In the meanwhile, we conduct bottleneck analysis for congestion at nodes or stairs respectively. We obtain the variation of number of people at nodes or stairs over time, and we use R and matlab to plot the curves of variation. Then we can tell the potential bottlenecks from the curves by analyzing the places where a great many of people gather for a long time. The result shows that bottlenecks **at nodes** tend to appear near stairs or exits, while the bottlenecks **at stairs or exits** seem to appear randomly.

To test the efficiency of our evacuation model, we compare our strategy with the simple straight-forward strategy, which means everyone rushes to the nearest exit or stairway. The result shows that the time cost of our strategy is 40% less than the simple strategy, which demonstrates our model's efficiency. Besides, we apply our model on another museum and get an ideal result. This implies the universality of our method.

Keywords: Undirected Graph, Floyd Algorithm, Multi-source Dynamic Network Flow, Improved Annealing Algorithm, Bottleneck Analysis

Contents

1 Introduction

1.1 Background

Recent years have witnessed a growth of terror attacks in the world. However, according to the survey[1], orderly evacuation during terror attacks or fire can greatly reduce casualties. Consequently, it is essential to design reasonable evacuation plans for popular scenic spots. With the aim of executing organized evacuation and minimizing the casualty during emergency, a system letting people get out of building in order with the minimal time cost is of vital significance.

Therefore, we have put forward a mathematics model based on actual structure of building and the real time distribution of people. The model is designed based on the Louvre, one of the world's largest and most visited art museum. In the meanwhile, our model is universally applicable, which means it can also be applied in other evacuation occasions.

In this paper, we focus on making plans for evacuation. For each floor, we divide the whole area into up to 16 areas, and allocate 3-5 stairways to visitors in each area. Then we combine the result of different floors and figure out the evacuation strategy for the whole building. In addition, we take different threats into consideration, such as fire, poison gas, and terrorist attacks. In these cases, some stairways may become unavailable and our strategy need to alter accordingly.

1.2 Problem restatement

The main problem is to develop an emergency evacuation model in order to evacuate visitors from the museum as quickly as possible. When we attempt to establish the model, we need to consider the limited capacity of stairs and find out people's evacuation paths from different areas. Therefore, we are required to solve the following problems:

- Determine the paths people should take from different starting points of departure.
- Determine the proportion of people taking each path from the same starting point.
- Identify potential bottlenecks that may limit movement towards the stairs or exits.

1.3 Related work

According to [1], macroscopic and microscopic models are two main types of models in evacuation problems. In macroscopic models, people are considered as homogeneous groups and have similar characteristics. In comparison, every one has their own choices and behaviours in microscopic models, thus some complex factors need to be considered when predicting their egress routes.

For macroscopic models, Hamacher and Tjandra [1] used **dynamic network flow** to describe evacuation problem; Campo et al. [2] employed **shortest path algorithm** to improve the efficiency of evacuation; Lu et al. [3] described the capcity constraints in

[1]http://www.sohu.com/a/252999487_100267724

evacuation problems using **time series analysis**. In general, graph theory and network flow method are the main methods employed in macroscopic models.

For microscopic ones, typical models include **social force model** [4], **cellular automaton model** [5], and **probabilistic model** [6].

1.4 Our work

The Louvre can be regarded as a complex network structure. In this case, the behaviours of individuals should not be paid much attention to, otherwise it will make the problem too complicated to solve. Therefore, we consider using **macroscopic models** to describe the evacuation plan of the Louvre.

Each floor of Louvre can be treated as an undirected graph, thus the problem can be converted to the multi-source dynamic network flow with capacity constraint. However, Sangho Kim and Shashi Shekhar [9] pointed out the NP completeness of this problem, which means it cannot be computed and solved in polynominal time. Therefore, we relax the capacity restrictions of edges and focus on the allocation of number of people from each starting point to each stair or exit, using improved annealing algorithm. Besides, we can easily identify potential bottlenecks in further analysis.

Our model remarkably outperforms the simple straight-forward strategy, as the total time cost for a single floor in our model is approximately 60% of the straignt-forward one. In the meanwhile, our model also successfully passes the universality test, thus proved to be pervasive.

2 Assumptions and notations

2.1 Assumptions

In order to facilitate the design of the model, we have determined several assumptions. These assumptions are based on actual situation.

- People will obey our arrangements in emergency.
- Stairways and exits are the main sources of congestion, and the time cost in these places is proportional to the number of people passing through.
- Exhibition rooms are big enough to hold any number of people, where we do not consider congestion at first. Despite congestion might occur at gates or narrow passageways which link exhibition rooms, we ignore their influence first when calculating the paths and will take them into account afterwards.
- Real time monitoring system frequently provides the number of people in each area. That is to say, when a terror attack or fire occurs, we have access to the distribution of people at every exhibition room immediately.

2.2 Notations

Here are some of the symbols we use in our model, which will be explained in detail in corresponding sections.

- T_{ij}^t : the total time for people in area i to reach the next floor through stairway j
- T_{ij}^f : the time for people in area i to reach the threshold of stairway j
- T_{pij}^t : the total time for people in previous area to reach next floor through stairway j
- N_i : the total number of people in area i
- λ_{ij} : the proportion of people in area i that will go to stairway j
- v_j : the speed of movement when people pass through stairway j
- $x_k l$: the number of people to be evacuated along the path P_l to the exit k
- K : the number of exits
- L : the number of paths
- T_{kl} : the time spent on the path P_l

3 Evacuation Model

3.1 Evacuation strategy for a single floor

In order to let all the visitors leave the Louvre safe and sound, we need to allocate evacuation paths for all of them. Since the building has multiple exits, the visitors should be allocated to different exits to minimize the possibility of congestion. And the proportion of allocation should be carefully calculated. According to [7], the problem can be classified as the Emergency evacuation problem for multi-exit buildings. To make the problem less complicated, We created a simplified structure of Louvre basing on the actual layout of the Louvre from the website[2](shown in Figure 1). The simplified structure we designed is shown in Figure 2.

For each floor in Louvre, we considered the whole floor as an **undirected graph**. Each exhibition room is considered as a **node** while gates or narrow passageways are considered as **edges**. People in each exhibition room are considered as to be gathered at the centre, so the middle points of every edge are exactly the starting points. Basing on the Assumption we made at section 2.1, the congestion is considered to occur only at stairs or exits, which implies that it takes little time for a group of people to pass through a single node. Before reaching the stairs or exits, time spent can be simply calculated as the distance divide the moving speed.

In the structure we designed, taking 1st floor as example, it is divided into **16 areas**, just like the edges displayed in figure 2, while **5 stairways** are the blue triangles in figure 2.

[2]https://https://www.louvre.fr/en/plan/

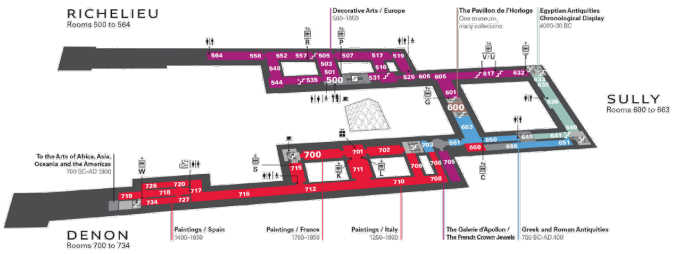

Figure 1: The actual structure of a single floor in Louvre

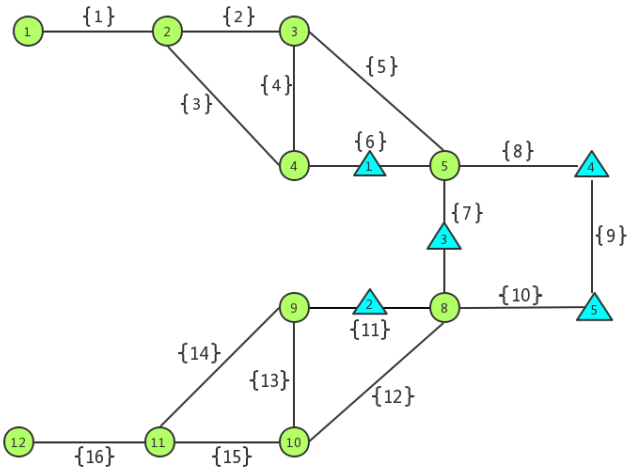

Figure 2: Simplified structure of a single floor in Louvre

3.1.1 The Calculation of Shortest paths

For each starting point to each stair or exit, only the shortest path is chosen.

First We employ the **classic Floyd Algorithm** to calculate a matrix $D_{n\times m}$, where D_{ij} is the length of shortest path between node i and j.

Then for each starting point s which lies in the middle of an edge $e = (s_1, s_2)$, if the destination also lies in the middle of an edge $e = (d_1, d_2)$, then we choose the the smallest value among $D_{s_1d_1}, D_{s_1d_2}, D_{s_2d_1}, D_{s_2d_2}$, else if the destination point is a single node d, naturally we compare D_{s_1d} and D_{s_2d} and choose the smaller one. Thus we obtain a time cost matrix $T_{e\times m}$ recording the time spent from each starting point to each stair or exit. In the matrix, the row label represents the label of a starting edge, while the column label shows the label of target stairway or exit.

As an example, the distance matrix $D_{n\times m}$ and the time cost matrix $T_{e\times m}$ for 2nd floor are shown at at table 1 and table 2 respectively.

During our calculation, the unit distance is set to be 50m, and the speed of pedestrians is considered 1.5 m/s.

Table 1: Distance matrix for 2nd floor

0	50	100	150	200	250	300	250	300	350	400	450
50	0	50	100	150	200	250	200	250	300	350	400
100	50	0	50	100	150	200	150	200	250	300	350
150	100	50	0	50	100	150	100	150	200	250	300
200	150	100	50	0	50	100	50	100	150	200	250
250	200	150	100	50	0	50	100	150	200	250	300
300	250	200	150	100	50	0	50	100	150	200	250
250	200	150	100	50	100	50	0	50	100	150	200
300	250	200	150	100	150	100	50	0	50	100	150
350	300	250	200	150	200	150	100	50	0	50	100
400	350	300	250	200	250	200	150	100	50	0	50
450	400	350	300	250	300	250	200	150	100	50	0

Table 2: Time cost matrix for 2nd floor

100	133	167	150	183
66.7	100	133	117	150
50	83.3	117	100	133
33.3	66.7	100	83.3	117
50	50	83.3	66.7	100
0	33.3	66.7	50	83.3
33.3	0	33.3	50	50
33.3	33.3	66.7	16.7	50
66.7	66.7	66.7	16.7	16.7
66.7	33.3	33.3	50	16.7
66.7	33.3	0	83.3	50
83.3	50	50	100	66.7
100	66.7	33.3	117	83.3
117	83.3	50	133	100
133	100	66.7	150	117
167	133	100	183	150

3.1.2 The Design of the Allocation model

To allocate the people in each exhibition room to different exits, the number of people and the total structure of the Louvre should be taken into the consideration. The proportion of the allocation is the most important parameter that influence the effectiveness of the evacuation. To find the ideal proportion, a mathematics model is built to help solving the problem.

The model is based on the **lemma** proved in [8]: If $1, 2, \cdots, \gamma$ is chosen as the exit for evacuation, then in the optimal evacuation scheme, the evacuation of all groups will complete at the same time: $T_1 = T_2 = \cdots = T_\gamma$

With the lemma above, we can then set up the mathematics model to solve the problem:

$$\min T(A) = \max T_{kl}(x_{kl}) \tag{1}$$

$$A = F(x, N_h, N_e, V_s, G(V, E)) \tag{2}$$

$$s.t. \sum_k \sum_l x_{kl} = x \tag{3}$$

$$x_{kl} \geq 0, k = 1, 2, \cdots, K, l = 1, 2, \cdots, L \tag{4}$$

- $x_k l$: the number of people to be evacuated along the path P_l to the exit k
- K : the number of exits
- L : the number of paths
- T_{kl} : the time spent on the path P_l
- x : the total number of people to be evacuated
- V_s : the evacuation speed of exit and stairways
- $G(N, E)$: the net to be evacuated, N stands for the Node, E stands for Edge

According to the assumptions in section 2.1, most congestion occurs in stairways and exits, and the time spent passing through the stairways(or exits) is proportional to the number of people. In comparison, we assume that the movement from areas to stairways is smooth and irrelevant with the number of people. Therefore, the minimum time to move from area i to next floor is

$$T_{ij}^t = T_{ij}^f + \frac{\lambda_{ij} N_i}{v_j} \tag{5}$$

- T_{ij}^t : the total time for people in area i to reach the next floor through stairway j
- T_{ij}^f : the time for people in area i to reach the threshold of stairway j
- N_i : the total number of people in area i
- λ_{ij} : the proportion of people in area i going to stairway j
- v_j : the speed of movement when people pass through stairway j

The equation (5) can be established only if the stairway is not congested when people from area i arrive at stairway, that is to say, people from previous area have already passed through the stairway. Otherwise, another equation has to be established

$$T_{ij}^t = T_{pij}^t + \frac{\lambda_{ij} N_i}{v_j} \tag{6}$$

- T_{pij}^t : total time for people in previous area to reach next floor through stairway j

Equation (7) shows that people start to pass through stairway when those in front of them have passed it.

3.1.3 The Calculation of the Allocation model

Based on equation (5) and (7), we can simulate sequential evacuation events. However, the calculation of the model above might be a problem. According to[7], the evacuation problem is NP-hard problem, which means that Polynomial algorithm cannot solve this kind of problem. Consequently, we employ **Improved Annealing Algorithm** to find the best evacuation plan. The flow chart of the algorithm is displayed in figure 3. Our goal is to calculate the proportion matrix $A(a_{ij})$. a_{ij} stands for the proportion of the people from the exhibition hall i to the exit j. And the proportion matrix $A(a_{ij})$ satisfy the following equation:

$$\sum_{m=1}^{j} a_{ij} = 1, i = 1, 2, 3, \cdots, N \tag{7}$$

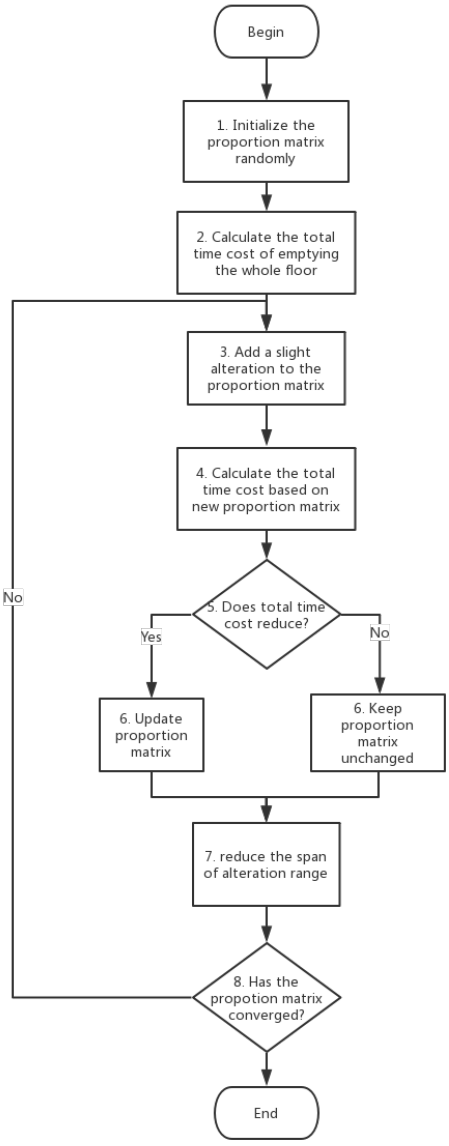

Figure 3: Flow chart of improved annealing algorithm

Though the equation 9 satisfies the lemma we presented in section3.1.2, to minimize the times of loop, we replace the equation 9 with the following function:

$$\min T(A) = c_1 \max T_{kl}(x_{kl}) + c_2(\max T_{kl}(x_{kl}) - \min T_{kl}(x_{kl})) \tag{8}$$

$$A = F(x, N_h, N_e, V_s, G(V, E)) \tag{9}$$

$$s.t. \sum_k \sum_l x_{kl} = x \tag{10}$$

$$x_{kl} \geq 0, k = 1, 2, \cdots, K, l = 1, 2, \cdots, L \tag{11}$$

- c_1 : Empirical value : 0.9
- c_2 : Empirical value : 0.1

After the modification, the loop we need to reach the accuracy of 0.01s drop from 194507 to 169206 times. Then We modify the Annealing Algorithm to make it more suitable for the condition. To begin with, the proportion of the allocation is generated randomly and calculate the corresponding time. Then we use the new proportion matrix to calculate the time needed for evacuation. If the time we need is smaller than the old one, the old matrix will be updated(Random Step1). In order to get the most ideal proportion with minimum times of loop, after several times of loop, we gradually reduce the range of random alteration(Random Step2). The algorithm will stop when the accuracy meet the demand. From figure 4 we can find that step 1 is used to get the Global Optimal Solution while the step 2 is used to promote the accuracy and minimize the times of loop.

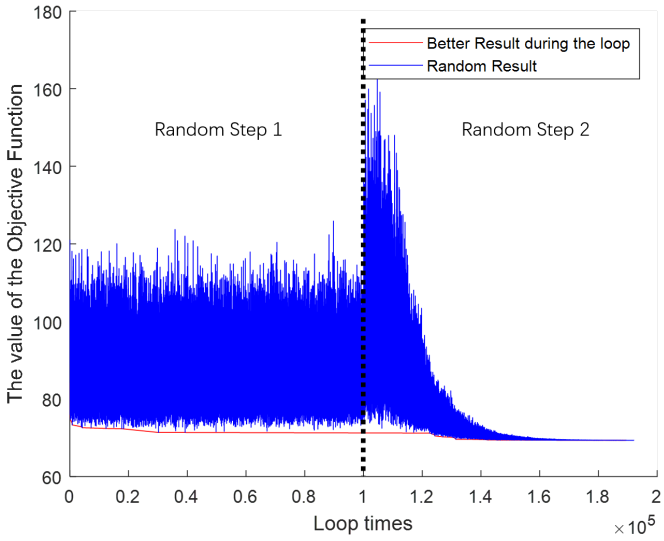

Figure 4: The value of the objective function during loop

To test the effectiveness of algorithm, we run the algorithm for several times and compare the results. Though the actual proportion is not the same every times, the time needed for evacuation remain constant with the variance of 0.27.

By using the algorithm, we can calculate Matrix of Allocation Proportion. The number of People in each exhibition hall is based on the analysis of the APP in the website

[3]. The figure5-8 shows the relationship between the times of loop and the time needed for the visitors in each floor to evacuate independently. From the figures we can find that with process of the algorithm, the evacuation time of each stair converge towards the same value. The time needed for each stairs is listed in the table3. Due to the unique stucture and function of B2 floor, we let all of the visitors to evacuate through the Pyramid Exit.

The detailed result of the allocation is listed in AppendixH

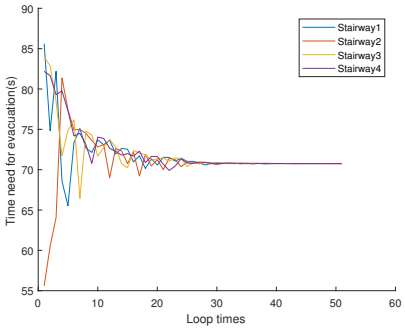

Figure 5: The Result of Evacuation Time in 2nd Floor During the Loop

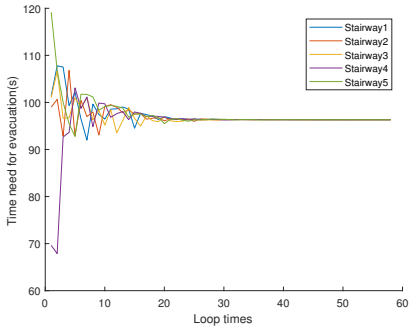

Figure 6: The Result of Evacuation Time in 1st Floor During the Loop

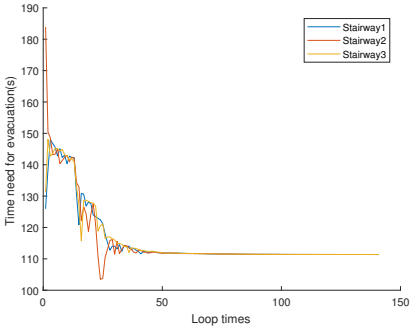

Figure 7: The Result of Evacuation Time in Ground Floor During the Loop

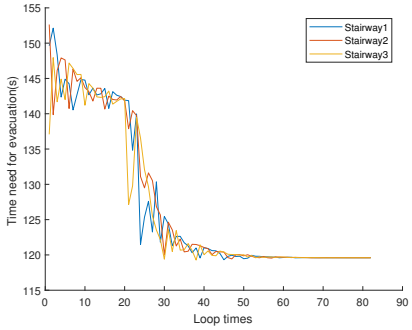

Figure 8: The Result of Evacuation Time in B1 Floor During the Loop

Table 3: The time(s) needed for evacuation

Floor\Exit	Exit1	Exit2	Exit3	Exit4	Exit5
2nd floor	70.75015	70.75137	70.75092	70.74989	
1st floor	96.31784	96.31938	96.31917	96.31996	96.31937
Ground floor	138.2667	138.2668	138.2667		
B1 floor	114.224	114.2254	114.2248		

[3]https://www.affluences.com/louvre.php

3.1.4 Single-floor Model in different Source of threats

In order to handle different kinds of threats and make relevant change, we make our model flexible and adaptable. When facing with different kinds of threats, such as fire, terror attacks, our model can change the control theory.

Table 4: The difference between fire and terror attack

	fire	terror attack
1	source of threat moves slowly	source of threat moves fast
2	require a smaller number of security	require a larger number of security
3	hard to control by the emergency personnel	easy to control by the emergency personnel

Basing on the function 2, the calculation of the allocation proportion matrix $A(a_ij)$ can be easily changed by the parameters in our model. If the fire causes the evacuation and blocks the essential paths, the shortest path calculated above will change, and $A(a_ij)$ will also change in accord with the path. Consequently, the allocation of people will also change and people will bypass the fire location. Comparing with the terror attack, the priority is the evacuation of people rather than the extinguish of fire. Only when all the people have left the building can emergency personnel extinguish the fire. In the mean time, people do not need a large number of security to protect them comparing with the terror attack, so the hidden path can be opened.

However, terror attack is totally a different condition, as the source of threats is moving fast. Consequently, people need to gather together to protect themselves, which means that Allocation is less important. And the hidden path should not be open because of the lack of security.

The condition to be changed while facing different source of threats are listed in table

Table 5: The difference between fire and terror attack

	fire	terror attack
1	Allocation of people is important	Allocation is less important
2	Hidden exits are open	Hidden exits are closed
3	Evacuation is the priority	Emergency personnel is the priority

3.2 Evacuation model for multiple floors

Basing on the single-floor evacuation model of Louvre we designed, we put forward the model considering the multiple floors. Since the passing speed of the stairways and exits are limited, we cannot consider the evacuation process independently, for the people in the 1st floor might stop the people in the 2nd from evacuating through the stairways. Thus, we need to consider the interaction between different floors.

In [10], the stairways of each floors are connected with each other, and the multi-storied building is considered as a huge undirected Graph, combining all the stairs and edges in a whole graph. The complexity of this algorithm increases dramatically with the increase of floors, which is not suitable for the calculation of high buildings.

Thanks to the lemma in section3.1.2, our model for the multi-storied building is simplified. Since the ideal allocation of people in a single floor makes all the stairways in the same floor synchronous, we may combine the different stairs in each floor together while solving the multi-storied Building. The solution is shown in figure 9-12.

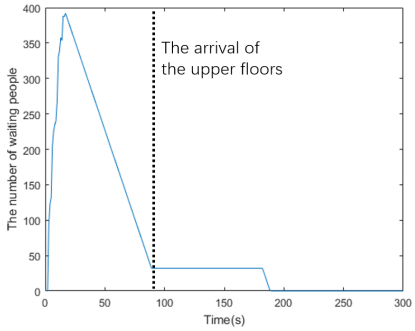

Figure 9: The number of waiting people at stair 1

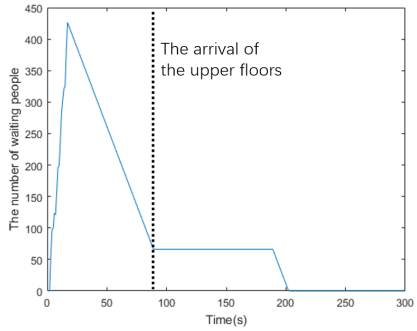

Figure 10: The number of waiting people at stair 2

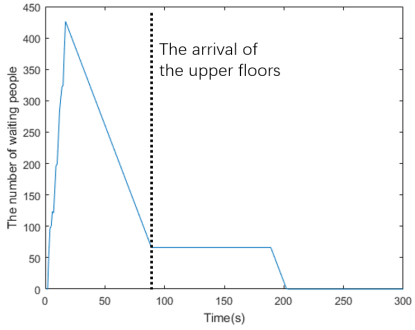

Figure 11: The number of waiting people at stair 3

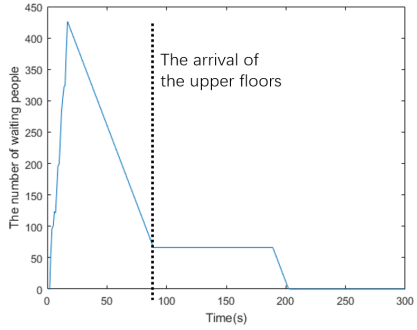

Figure 12: The number of waiting people at stair 4

3.2.1 Multi-floor Model in different Source of threats

The model is also influenced by the source of the threats. Consequently, the multi-storied evacuation model can be altered in accord with the threat source. However, unlike the single-floor model, the multi-floor model don't have a lot of parameter to take into consideration. The Priority level of each floor should be altered basing on the source of threats.

Table 6: The difference between fire and terror attack

	fire	terror attack
1	Top layers should be evacuate first	Underground layers should be evacuate first

Basing on the analysis above, the Flow chart of the whole evacuation plan is shown in the appendix I.

3.3 The entry of emergency personnel

The entry of emergency personnel is considered separately while facing fire and terror attack, which is based on the actual fact. Facing fire in the building, the evacuation is the top priority, while the entry of emergency personnel is considered as the most important thing facing terror attacks. The following figure shows that the model can handle the situation and spare the room for the emergency personnel to come in.

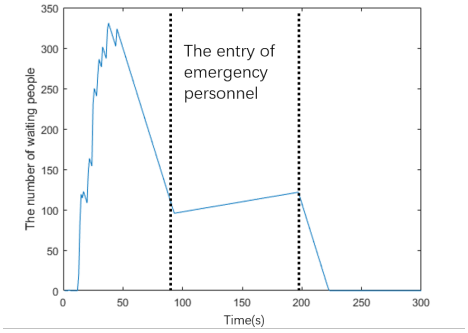

Figure 13: The number of waiting people with the entry of emergency personnel

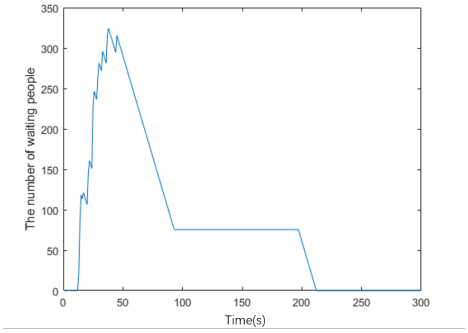

Figure 14: The number of waiting people without the entry of emergency personnel

4 Identification of potential bottlenecks

4.1 Bottlenecks at nodes

We ignore the influence of possible congestion at gates or narrow passageways when calculating the paths and work out the allocation. Now since we need to explore the possible bottlenecks at nodes, we investigate the change of number of people at each node over time. Still we take 1st floor as example. The corresponding plot of each node is presented in Figure 15. The curves are fitted using the automatic smooth function of the package ggplot2 in R.

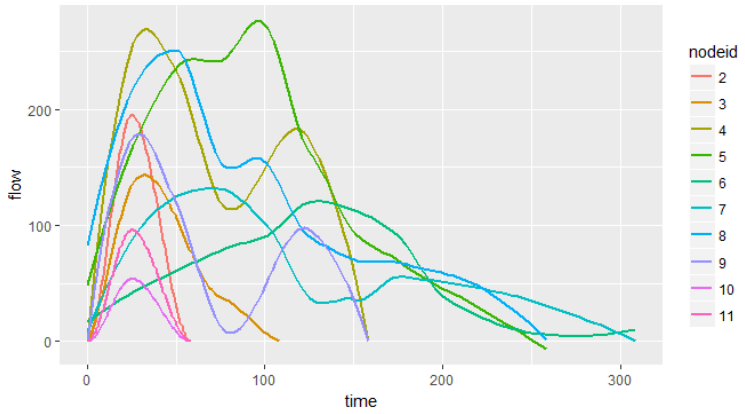

Figure 15: Flows of nodes on 1st floor

From the above plot, we can find that the continuous gathering of people mostly occurs at node 6, 7, namely the **bottlenecks**. Referring to figure 2, it is obvious that these two nodes are rather close to stairs on 1st floor. This phenomenon conforms to our intuition that congestion is more likely to occur near stairs. This implies that node 6 and 7 on 1st floor and other nodes nearest to stairs or exits should be paid more attention to.

4.2 Bottlenecks at stairs

The judgement of the bottlenecks at stairs is achieved by comparing the maximum value of waiting people, basing on the multi-floor model. The following figure 16-19 shows the number of waiting people of each stairway in the 1st floor, of which the stairways and exits are also sources of people.

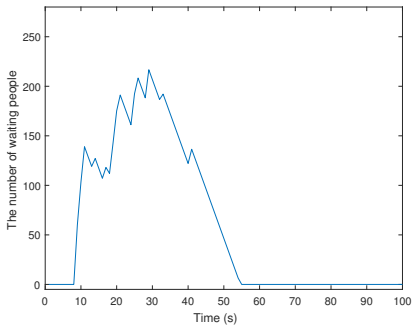

Figure 16: The number of waiting people at stair 1

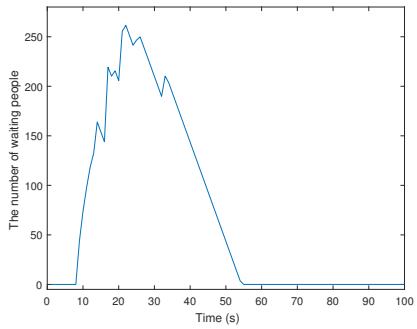

Figure 17: The number of waiting people at stair 2

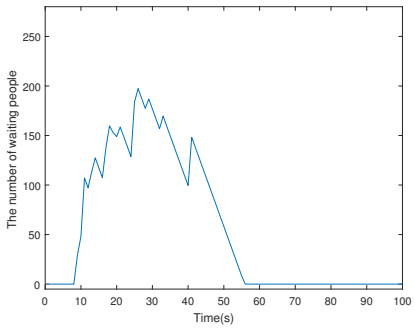

Figure 18: The number of waiting people at stair 3

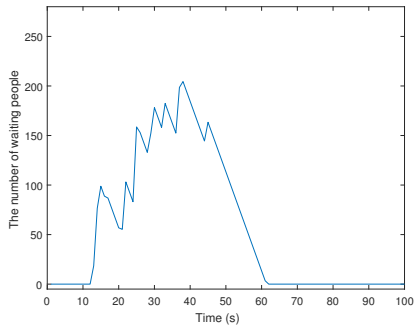

Figure 19: The number of waiting people at stair 4

By observing the figure above, we find that stair 2 is the Bottleneck of the system. The maximum number will decrease in a large scale if V_s increases.

5 Sensitivity analysis

In the model above, the time needed for evacuation is strongly related with the allocation proportion matrix $A(a_{ij})$. In addition, it is also related with the number of people and the structure of Louvre. To test the robustness of the model. the parameter is changed to test the effectiveness of the model.

Table 7: Sensitivity analysis

people	3754	3239	2753	2252	2002	1752
time(s)	171	155.549	146.3799	133.0425	126.4728	117.6087
Duty Cycle(%)	0.8636	0.8547	0.8369	0.7154	0.6359	0.5606

6 Comparison with simple straight-forward strategy

For a single floor, when terror attacks or fire occurs, it is easy to think of the simple straight-forward strategy: People at each starting point simply choose the nearest stair or exit. That is to say, in the allocation matrix, there is only one non-zero element in each row, which equals to the number of people in the corresponding staring point. Next we compare this simple strategy with the strategy we propose from several perspectives.

6.1 Congestion at nodes

Similar to our analysis in section 4.1, we plot the change of number of people at each node over time. We take the ground floor as example, and the two plots are as follows:

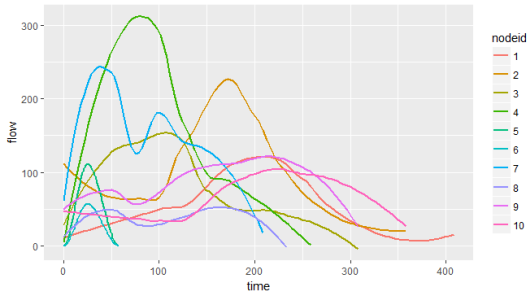

Figure 20: Flows of nodes on Ground floor using advanced strategy

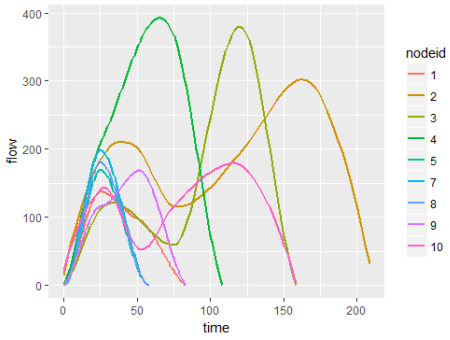

Figure 21: Flows of nodes on Ground floor using simple strategy

We notice that the peak values of the former plot are lower and peak number is fewer than the latter. Besides, it is obvious that the average flow of people at nodes during the process of evacuation is denser than the latter. It suggests that congestion will become severer if we choose the simple straight-forward strategy for evacuation, which proves the efficiency of our strategy. Though it seems that the time spent on the way to stairs or exits in simple strategy is shorter, we actually focus on the total time cost, including the time people spent going downstairs or through exits. For the total time cost, our strategy is much more efficient.

6.2 Congestion at exits

After comparing the congestion at nodes, we then focus on the exits and stairways. By comparing the congestion with and without the model, we can judge the effectiveness of the model. The result is shown in Figure22-25. Without the allocation of the model, people will choose the nearest exit, which will result in the underutilization of all the exits.

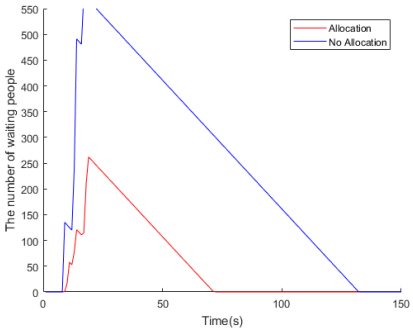

Figure 22: The Waiting number with and without the Allocation in Stairway 1

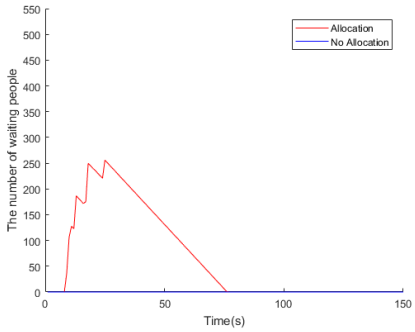

Figure 23: The Waiting number with and without the Allocation in Stairway 2

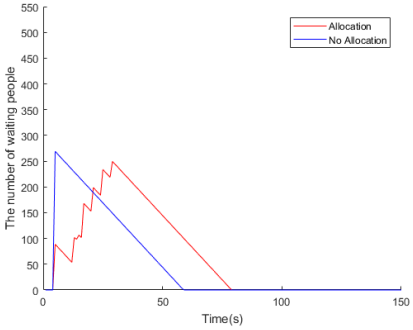

Figure 24: The Waiting number with and without the Allocation in Stairway 3

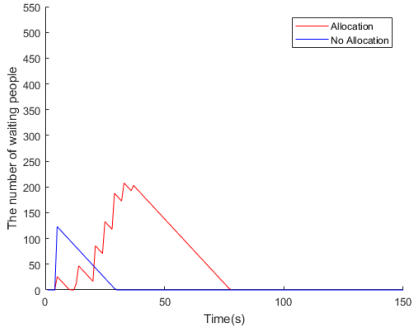

Figure 25: The Waiting number with and without the Allocation in Stairway 4

7 Universality

In order to test the universality of our evacuation model, we apply our method to another example. Figure 26 shows the planar graph of Shanhaiguan Great Wall Museum, which we transform into an undirected graph.

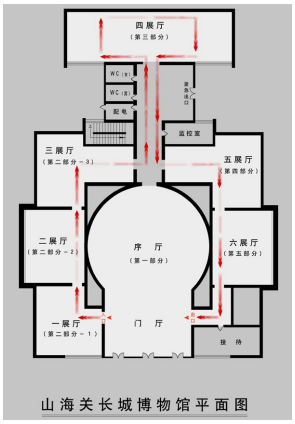

Figure 26: Planar graph of Shanhaiguan Great Wall Museum

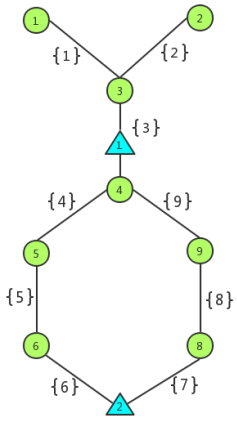

Figure 27: Simplified structure of Shanhaiguan Great Wall Museum

The change of flows at nodes over time is shown in Figure 28. We can tell from the plot that node 4 is the bottleneck. Referring to Figure 26, node 4 is the only crossroad at the center of the planar graph.

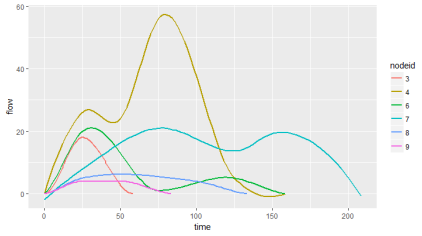

Figure 28: Flows of nodes using advanced strategy

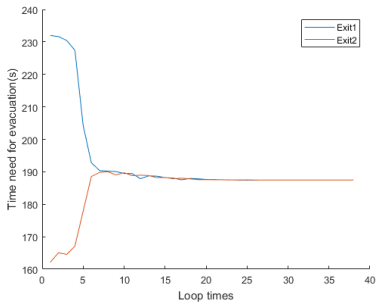

Figure 29: The result of evacuation time during the loop

The evacuation time can be calculated using the same method, which is shown in Figure29. The calculating process and result shows the universality of out model.

8 Recommendations for emergency management

- According to our model, the evacuation process is step by step, and different floors should evacuate in order. On such occasions, people need to obey our arrangements and reduce unpredictable behaviours. Therefore, many guards are required in each floor to keep the order and provide enough guidance. So we think the Louvre should have the ability to send enough guards when emergency occurs. In addition, there should be enough video or audio facilities in the building that remind people of the directions of each stairway and exit.

- Our calculation implies that the total evacuation time almost multiplies as floor number increases, which means that it takes much more time for people on the 2nd floor to get out of the Louvre. Therefore, we recommend that the 2nd floor should hold fewer people so that total evacuation time may reduce when emergency occurs. We can achieve such target by reducing the amount of exhibitions on the 2nd floor or building direct express lift or stairs from 2nd directly towards outside.
- The bottleneck analysis indicates that the exhibition rooms nearer to stairs or exits are more likely to congest. So we suggest that the gates or passageways near stairs or exits should be made more spacious and secret exits far away from current public ones should be opened up for the benefit of evacuation.

9 Strengths and Weaknesses

9.1 Strengths

- **Save computation cost**
  The computation complexity of Floyd algorithm is $O(N^3)$ where N is the number of nodes of an undirected graph. Since N is not very large, the computation does not cost much. The computation complexity of the improved annealing algorithm is controlled by our setting, which usually does not take much time either. Therefore, our model saves computation cost compared with those methods using simulation like cellular automaton. This is extremely important because when threats occur, time is life.
- **Efficient in saving the total time cost**
  As stated before, our method outperforms the simple straight-forward strategy in time cost by 40%, which is significant progress. It is vital to minimize the total evacuation time in emergency.

9.2 Weaknesses

- **Ignore people's variety**
  Our model assumes the speed of everyone is the same. To simplify our model, we do not take the disabled or the elderly into consideration.
- **Neglect the capacity constraint of edges and nodes**
  In our model, we trade off neglecting the capacity constraint of edges and nodes against the simplification of the algorithm to save computation. Despite it working well, in extreme cases there might be severe congestion in edges or nodes, which our model cannot prevent.

References

[1] Hamacher H., Tjandra S.(2001). Mathematical modeling of evacuation problems: State of the art. Pedestrian & Evacuation Dynamics.

[2] Campos, V.(2000). Evacuation transportation planning : a method of identify optimal independent routes. Urban Transport V, 3, 555-564.

[3] Lu, Q., George, B., & Shekhar, S.(2005). Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. International Symposium on Spatial & Temporal Databases. Springer, Berlin, Heidelberg.

[4] Helbing, D., Farkas, I. J., & Vicsek, T.(2000). Simulating dynamical features of escape panic. Social Science Electronic Publishing, 407(6803), 487-90.

[5] Kirchner, A., & Schadschneider, A.(2002). Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. Physica A Statistical Mechanics & Its Applications, 312(1), 260-276.

[6] Farahmand, M., Garetto, C., Bellotti, E., Brennan, K., Goano, M., & Ghillino, E., et al. (2001). Monte carlo simulation of electron transport in the iii-nitride wurtzite phase materials system: binaries and ternaries. IEEE Transactions on Electron Devices, 48(3), 535-542.

[7] Jianghua,Z.,Zhiping,L.,Daoli,Z.(2009).Emergency evacuation model and algorithm for multi-source sudden disaster accidents[J].Journal of Management Science,2009,(03):111-118.

[8] Chen P,Feng F. A fast flow control algorithm for real-time emergency evacuation in large indoor areas[J]. Fire Safety Journal, 2009,44(5):732-740

[9] Kim, S. , & Shekhar, S. . (2005). Contraflow network reconfiguration for evacuation planning: a summary of results. Acm International Workshop on Geographic Information Systems. ACM.

[10] Yang,J., Gao,Y.,Wang,H.(2014).Multi-storied Building Emergency Evacuation Model and Algorithm[J].Journal of System Simulation,2014,26(02):267-273.

Appendices

Appendix A Graph structure of each floor of the Louvre

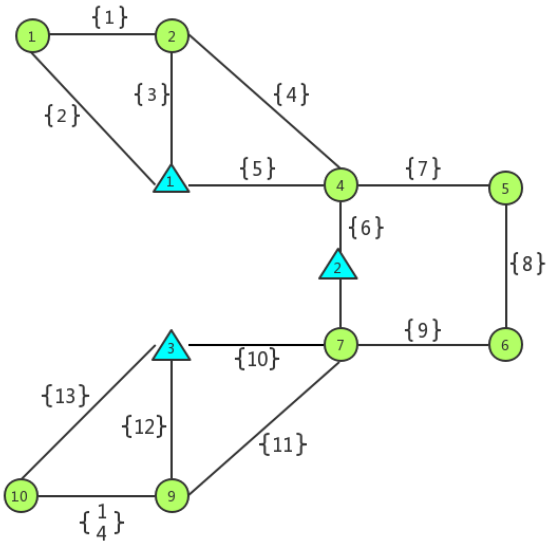

Figure 30: graph structure of floor B1

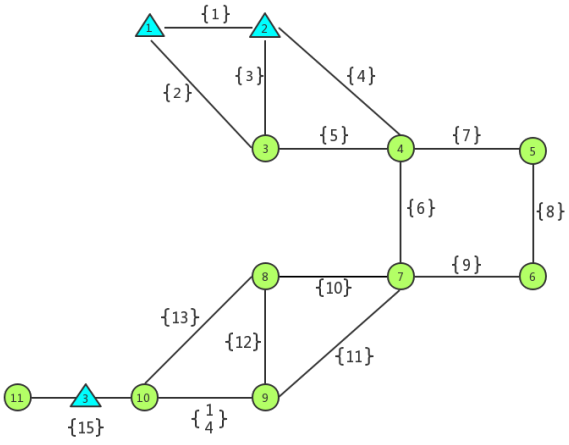

Figure 31: graph structure of floor 0

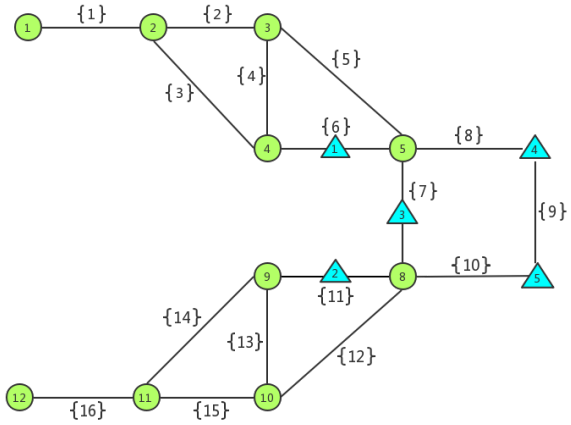

Figure 32: graph structure of floor 1

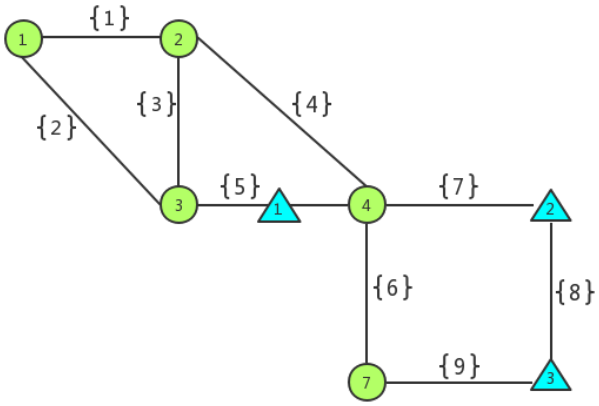

Figure 33: graph structure of floor 2

Appendix B Code for finding shortest paths and get node flows

The code is used for python 3.6

```
class Edge():
    def __init__(self, vi = -1, vo = -1, w = -1):
        self.Vi = vi
        self.Vo = vo
        self.W = w
class Dest():
    def __init__(self, kind = "e", id = -1):
        self.kind = kind
        self.id = id

import os
import numpy as np
import random
import matplotlib.pyplot as plt

##set unit length and speed of pedestrians
distper = 50
speed = 1.5
#os.chdir("your directory")


##calculate the time cost from a starting point at edge e to stair d
def cost(e, d):
    if(dests[d].kind == "e"):
        if(e == dests[d].id):
            return 0
        e1 = edges[e]
        e2 = edges[dests[d].id]
        return (min(dist[e1.Vi, e2.Vi], dist[e1.Vi, e2.Vo], dist[e1.Vo, e2.Vi], dist[e1.Vo, e2.
    else:
        e = edges[e]
        v = dests[d].id
        return (min(dist[e.Vi, v], dist[e.Vo, v]) + 0.5*e.W)/speed


##get the shortest path between node s and node d
def searchPath(s, d):
    path = [s,]
    v = s
    while(v != d):
        random.shuffle(neibs[v])
        for vt in neibs[v]:
            if(dist[vt, d] + dist[v, vt] == dist[v, d]):
                v = vt
                path.append(v)
                break
    return path

def myplot(i):
    plt.plot(flows[i][0], flows[i][1])

infty = 10000

##read in the graph, calculate the time cost matrix for the matlab code to conduct annealing al
def graph(floor):
```

```
    global edges, dests, dist, cost_mat, n, ne, m, neibs
    edges = [Edge(),]
    dests = [Dest(),]
    ##read in the graph.
    ##The three columns are respectively Vi, Vo, W
    f = open("graph" + str(floor) + ".txt", 'r')
    lines = f.readlines()
    for line in lines:
        line = line.replace("\n", "")
        temp = line.split(' ')
        e = Edge(int(temp[0]), int(temp[1]), int(temp[2])*distper)
        edges.append(e)
    f.close()
    n = max(max(list(map(lambda x: x.Vi, edges))), max(list(map(lambda x: x.Vo, edges))))
    ne = len(edges)-1
    ##read in stairs or exits.
    ##The two colums are respectively the type(on an edge or at a node) and the id
    f = open("exits" + str(floor) + ".txt", 'r')
    lines = f.readlines()
    for line in lines:
        line = line.replace("\n", "")
        temp = line.split(" ")
        dests.append(Dest(temp[0], int(temp[1])))
    f.close()
    m = len(dests)-1
    ##Initialize the distance matrix
    dist = np.zeros((n+1, n+1))
    dist = dist + infty
    for i in range(n+1):
        dist[i, i] = 0
    for e in range(1, ne+1):
        dist[edges[e].Vi, edges[e].Vo] = edges[e].W
        dist[edges[e].Vo, edges[e].Vi] = edges[e].W
    ##floyd algorithm
    for k in range(1, n+1):
        for i in range(1, n+1):
            for j in range(1, n+1):
                dist[i, j] = min(dist[i, j], dist[i, k] + dist[k, j])
    ##calculate cost matrix
    cost_mat = np.zeros((ne+1, m+1))
    for i in range(1, ne+1):
        for j in range(1, m+1):
            cost_mat[i, j] = cost(i, j)
    ##write out cost matrix
    f = open("costs" + str(floor) + ".txt", "w")
    for i in range(1, ne+1):
        for j in range(1, m+1):
            f.write(str(cost_mat[i, j]) + ' ')
        f.write("\n")
    f.close()
    ##establish the adjacency list for every node
    neibs = list(range(n+1))
    for i in range(len(neibs)):
        neibs[i] = []
    for e in range(1, ne+1):
        neibs[edges[e].Vi].append(edges[e].Vo)
        neibs[edges[e].Vo].append(edges[e].Vi)


##get the shortest path between any two node
def getpaths():
```

```
    global paths
    paths = [[],]
    for e in range(1, ne+1):
        path = [[],]
        for d in range(1, m+1):
            if(dests[d].kind == "e"):
                if(dests[d].id == e):
                    path.append([])
                    continue
                e1 = edges[e]
                e2 = edges[dests[d].id]
                v1c = [e1.Vi, e1.Vo]
                v2c = [e2.Vi, e2.Vo]
                temp = infty*ne
                for v1 in v1c:
                    for v2 in v2c:
                        if(dist[v1, v2] < temp):
                            v1r = v1
                            v2r = v2
                            temp = dist[v1, v2]
                path.append(searchPath(v1r, v2r))
            else:
                if(dist[edges[e].Vi, dests[d].id] < dist[edges[e].Vo, dests[d].id]):
                    vr = edges[e].Vi
                else:
                    vr = edges[e].Vo
                path.append(searchPath(vr, dests[d].id))
        paths.append(path)

##read in the information of allocation and people from the result of annealing algorithm
def getalloc(floor):
    global palloc
    ##read in people
    f = open("people" + str(floor) + ".txt", "r")
    lines = f.readlines()
    people = np.zeros((ne+1, 1))
    for i in range(1, ne+1):
        lines[i-1] = lines[i-1].replace("\n", "")
        people[i, 0] = int(lines[i-1])
    f.close()
    ##read in the proportion of allocation
    f = open("allocation" + str(floor) + ".txt", "r")
    lines = f.readlines()
    alloc = np.zeros((ne+1, m+1))
    for i in range(1, ne+1):
        lines[i-1] = lines[i-1].replace("\n", "")
        temp = lines[i-1].split("\t")
        for j in range(1, m+1):
            alloc[i][j] = temp[j-1]
    f.close()
    ##get the allocation matrix
    palloc = people*alloc

##get the flow change over time for each node
def getflow():
    global nodeflows, flows
    for i in range(1, ne+1):
        for j in range(1, m+1):
            palloc[i, j] = int(palloc[i, j])
    nodeflows = list(range(n+1))
    for i in range(n+1):
```

```
        nodeflows[i] = dict()
    for i in range(1, ne+1):
        for j in range(1, m+1):
            if(paths[i][j] == []):
                continue
            s = paths[i][j][0]
            for v in paths[i][j]:
                t = edges[i].W/2 + dist[s, v]
                if(str(t) in nodeflows[v]):
                    nodeflows[v][str(t)] += palloc[i, j]
                elif(palloc[i, j]!=0):
                    nodeflows[v][str(t)] = palloc[i, j]
    sortnf = list(range(n+1))
    for i in range(n+1):
        sortnf[i] = sorted(nodeflows[i].items(), key = lambda x: float(x[0]))
    flows = list(range(n+1))
    for i in range(n+1):
        flows[i] = [list(map(lambda x: float(x[0]), sortnf[i])), list(map(lambda x: x[1], sortn



##write out node flows for R code to draw plots
def dumpflow(floor):
    f = open("flow" + str(floor) + ".csv", "w")
    f.write("time,flow,nodeid\n")
    for i in range(1, n+1):
        f.write("0,0,"+str(i)+"\n")
        l = len(flows[i][0])
        if(l == 0):
            continue
        for j in range(l):
            if(flows[i][1][j] != 0):
                f.write(str(flows[i][0][j]) + "," + str(flows[i][1][j]) + "," + str(i) + "\n")
        f.write(str(flows[i][0][l-1] + distper/speed) + ",0," + str(i) + "\n")
    f.close()

##Allocate using straight-forward strategy
def straightalloc(floor):
    global palloc
    ##read in people
    f = open("people" + str(floor) + ".txt", "r")
    lines = f.readlines()
    people = np.zeros((ne+1, 1))
    for i in range(1, ne+1):
        lines[i-1] = lines[i-1].replace("\n", "")
        people[i, 0] = int(lines[i-1])
    f.close()
    ##get allocation matrix directly base on cost matrix
    palloc = np.zeros((ne+1, m+1))
    for i in range(1, ne+1):
        tempmin = infty
        index = -1
        for j in range(1, m+1):
            if(cost_mat[i][j] < tempmin):
                tempmin = cost_mat[i][j]
                index = j
        palloc[i][index] = people[i]


for floor in range(1, 5):
    graph(floor)
```

```
for floor in range(1, 5):
    graph(floor)
    getpaths()
    getalloc(floor)
    getflow()
    dumpflow(floor)
```

Appendix C Code for plotting the node flow curves

The code is used for R 3.4.3

```
library(ggplot2)
raw = read.csv("flow-x.csv", stringsAsFactors = F)
raw$nodeid = as.factor(raw$nodeid)
m = ggplot(data = raw, aes(x = time, y = flow), group = nodeid)
##The smooth curve
m + stat_smooth(size = 1, aes(color = nodeid), se = F)

##The broken line
#m + geom_line(aes(color = nodeid), size = 1) + geom_point(aes(color = nodeid))
```

Appendix D Code for calculate the Allocation proportion matrix

The code is used for Matlab 2017a

```
function [Allocation,Result,Duty,Start_time,Finish_time,Wait_num,People_get,Wait_num_uc] = Anne
Loop_max = 100000;
a = 0.9999;
t0 = 120; %the begin temp
tf = 1; %the final temp
t = t0; %initial the temp
Speed = 5; %the speed of running
[exhibit_num,stair_num] = size(Distance);
%People_num = 100.*rand(exhibit_num,1); %the num of people,to be loaded
%People_num = textread('people.txt','%d');
%People_num =People_num* 100;
Allocation = rand(exhibit_num,stair_num); %initial the allocation
Allocation_sum = sum(Allocation');
%[data1,data2,data3,data4,data5] = textread('costs.txt','%f %f %f %f %f');
%Distance = [data1,data2,data3,data4,data5];
%Distance = 10.*Distance;
%Distance(:,5) = Distance(:,5) .* 100;
%T_need = zeros(exhibit_num,stair_num); %initial the T we need
T_way = Distance./Speed; %the time spent on the way
effect_T = zeros(stair_num,2);
Speed_A = 5;
Speed_stair = Speed_A*ones(stair_num,1); %initial the speed of stairs, to be loaded
T_reg = zeros(stair_num,500);
T_reg_times = 0;
Greedy_domain = 1;
greedy_flag = 0;
goal = zeros(Loop_max,1);
goal_reg = ones(300,1);
Fig_Nomoal = zeros(2*Loop_max,2);
Fig_Good = zeros(2*Loop_max,2);
%% pre process
%unit
for i = 1:exhibit_num
    Allocation(i,1:stair_num) = Allocation(i,1:stair_num)/Allocation_sum(i);
end
%sort the t
[~,T_way_index] = sort(T_way); %ţÚijÿÐąţÄÔÚÔ■Àt'¿ØÕóţÄţÚijÿžÅ
%% Anneal
Loop_count = 0;
while Greedy_domain > 0.0001
    Loop_count = Loop_count + 1;
    %% šúÉúÈÅűŕ
    %Allocation_new = Allocation + rand(exhibit_num,stair_num)*(1-greedy_flag)+greedy_flag*(2.*
    Allocation_new = Allocation + rand(exhibit_num,stair_num)*(1-greedy_flag)+greedy_flag*(2.*r
    Allocation_new = abs(Allocation_new);
    Allocation_new_sum = sum(Allocation_new');
    for i = 1:exhibit_num %unit
        Allocation_new(i,1:stair_num) = Allocation_new(i,1:stair_num)/Allocation_new_sum(i);
    end
    %% ijÆËã¡áźû
    for i = 1:stair_num
        effect_T(i,1) = Calculate(Allocation_new(:,i),T_way(:,i),T_way_index(:,i),People_num,Sp
        effect_T(i,2) = Calculate(Allocation(:,i),T_way(:,i),T_way_index(:,i),People_num,Speed_
```

```
        %[effect_T(i,1),effect_T(i,2)] = [Calculate(Allocation_new(:,i),T_way(:,i),T_way_index(
    end
    %% śÈ¡ÏÓët'ęÀí
    eita = 0.99;
    goal(Loop_count) = eita*max(effect_T(:,1))-(1-eita)*min(effect_T(:,1))-eita*max(effect_T(:,
    %goal(Loop_count) = (max(effect_T(:,1))-min(effect_T(:,1))-max(effect_T(:,2))+min(effect_T(
    Fig_Nomoal(Loop_count,1) = eita*max(effect_T(:,1))-(1-eita)*min(effect_T(:,1));
    Fig_Nomoal(Loop_count,2) = Loop_count;
    if goal(Loop_count) < 0
        Allocation = Allocation_new;
        T_reg_times = T_reg_times+1;
        T_reg(:,T_reg_times) = effect_T(:,1);
        goal_reg(T_reg_times) = goal(Loop_count);
        Fig_Good(T_reg_times,1) = eita*max(effect_T(:,1))-(1-eita)*min(effect_T(:,1));
        Fig_Good(T_reg_times,2) = Loop_count;
%         if error_reg(T_reg_times)<1
%             break;
%         end
    end
    if Loop_count>Loop_max
        t = t * a;
        Greedy_domain = Greedy_domain*a;
        greedy_flag = 1;
    end
end
figure(1)
plot(goal_reg(1:T_reg_times));
Totaltime = max(T_reg(:,T_reg_times));
%Duty = (sum(People_num)/sum(Speed_stair))/(Totaltime-min(min(T_way)));
figure(2)
hold on;
for i =1:stair_num
    plot(1:T_reg_times,T_reg(i,1:T_reg_times));
end
legend('Stairway1','Stairway2','Stairway3','Stairway4','Stairway5');
xlabel('Loop times'), ylabel('Time need for evacuation(s)');
hold off
figure(3)
hold on;
plot(Fig_Good(1:T_reg_times,2),Fig_Good(1:T_reg_times,1),'r');
plot(Fig_Nomoal(1:Loop_count,2),Fig_Nomoal(1:Loop_count,1),'b');
xlabel('Loop times'), ylabel('The value of the Objective Function');
legend('Better Result during the loop','Random Result');
hold off
Result = T_reg(:,T_reg_times);
%% get the wait time
Time_count_max = 300;
Wait_num = zeros(Time_count_max,stair_num);%ceil(max(Result)));
Time_max = Time_count_max;%ceil(max(Result));
Finish_time = zeros(stair_num,1);
Start_time = zeros(stair_num,1);
%People_get = zeros(Time_max,stair_num);
for i = 1:stair_num
    %People = People_num.*Allocation(:,i);
    [Wait_num(1:Time_max,i),Start_time(i),Finish_time(i),People_get(:,i)] = Get_Waitnum(People_
    figure(3+i);
    plot(People_get(1:70,i));
end
Duty = (sum(People_num)/sum(Speed_stair))/(max(Finish_time)-min(Start_time));
%% compare with the uncontroled condition
Finish_time_uc = zeros(stair_num,1);
```

```
Start_time_uc = zeros(stair_num,1);
Wait_num_uc = zeros(Time_count_max,stair_num);%ceil(max(Result)));
Allocation_uc = zeros(exhibit_num,stair_num);
%sort the t
[~,T_way_index2] = sort(T_way'); %ţÚijÿÐąţÄÔÚÔ■Àt'¿ØÕóţÄţÚijÿžÅ
T_way_index2 = T_way_index2';
for i = 1:exhibit_num
    Allocation_uc(i,T_way_index2(i,1)) = 1;
end
for i = 1:stair_num
    [Wait_num_uc(1:Time_max,i),Start_time_uc(i),Finish_time_uc(i),People_get_uc(:,i)] = Get_Wai
end
end
```

```
function [T] = Calculate(new,T_way,T_way_index,People_num,Speed_stair)
People = new.*People_num;
L = length(People);
%% process the data
T = T_way(T_way_index(1)) + People(T_way_index(1))/Speed_stair;%modified
exit_flag = 0;
for i = L:-1:10
    if People(T_way_index(i)) <1
        exit_flag = exit_flag+1;
    else
        break;
    end
end
for i =1:(L-1-exit_flag)
   if T_way(T_way_index(i+1))> T %empty
       T = T_way(T_way_index(i+1))+People(i+1)/Speed_stair;
   else %not empty
       T = T + People(i+1)/Speed_stair;
   end
    %T = T + max(T_way(T_way_index(i+1))-T_way(T_way_index(i)),People(T_way_index(i))/Speed_sta

end
flag = (T>sum(People));
%assert(flag == 1,'error1')
end
```

Appendix E Code for calculate the Allocation proportion matrix

The code is used for Matlab 2017a

```
function [T] = Calculate(new,T_way,T_way_index,People_num,Speed_stair)
People = new.*People_num;
L = length(People);
%% process the data
T = T_way(T_way_index(1)) + People(T_way_index(1))/Speed_stair;%modified
exit_flag = 0;
for i = L:-1:10
    if People(T_way_index(i)) <1
        exit_flag = exit_flag+1;
    else
        break;
    end
```

```
end
for i =1:(L-1-exit_flag)
   if T_way(T_way_index(i+1))> T %empty
       T = T_way(T_way_index(i+1))+People(i+1)/Speed_stair;
   else %not empty
       T = T + People(i+1)/Speed_stair;
   end
    %T = T + max(T_way(T_way_index(i+1))-T_way(T_way_index(i)),People(T_way_index(i))/Speed_sta

end
flag = (T>sum(People));
%assert(flag == 1,'error1')
end
```

Appendix F Code for calculate the number of waiting people

The code is used for Matlab 2017a

```
function [ Wait_num_store,Start_time,Finish_time,People_get] = Get_Waitnum( People_num,Allocati
exhibit_num = length(Allocation);
People = People_num.*Allocation;
Wait_index = -ones(exhibit_num,1);%initialize
Wait_time_index = zeros(exhibit_num,1);
Wait_num = 0;
People_get = zeros(Time_max,1);
Wait_num_store = zeros(Time_max,1);
T = T_way(T_way_index(1)) + People(T_way_index(1))/Speed_stair;%modified
for i =1:(exhibit_num-1)
    if T_way(T_way_index(i+1))> T %empty
        T = T_way(T_way_index(i+1))+People(i+1)/Speed_stair;
        Wait_index(T_way_index(i+1)) = 0;
        %Wait_time_index(T_way_index(i+1)) = T_way(T_way_index(i+1));
    else %not empty
        T = T + People(T_way_index(i+1))/Speed_stair;
        Wait_index(T_way_index(i+1)) = 1;
        Wait_time_index(T_way_index(i+1)) = T_way(T_way_index(i+1));
    end
end
loop_time = 1;
Start_time = 0;
flag_once = 1;
Finish_time = 0;
for time =2:Time_max%(Time_max+10+sum(People))
    if Wait_num ~=0 && Start_time==0
        Start_time = time;
    end
    if Wait_num >0
        Wait_num = Wait_num - Speed_stair;
        Finish_time = time;
    end
    if Wait_num<0
        Wait_num = 0;
    end
    if (flag_once == 1) && (time >T_way(T_way_index(1)))
        People_get(time) = People_get(time-1) + People(T_way_index(1));
        flag_once = 0;
```

```
    end
    People_get(time) =  People_get(time-1);
    if (time > T_way(T_way_index(loop_time+1)))&&(loop_time<exhibit_num)&&Wait_index(T_way_inde
        Wait_num = Wait_num + People(T_way_index(loop_time+1));
        People_get(time) =  People_get(time-1) + People(T_way_index(loop_time+1))*sign(Wait_num
        Wait_index(T_way_index(loop_time+1)) = -1;
        loop_time = loop_time + 1;
        if loop_time == exhibit_num
            loop_time = exhibit_num-1;
        end
    end
    Wait_num_store(time) = Wait_num;
end
end
```

Appendix G Code for calculate multilevel allocation

The code is used for Matlab 2017a

```
clear,clf;
%% multilevel initial
stair_num = [5,5,5,5,3];%ËşÐò2,1,0čň-1čň-2čňÌÓÉúů¡Ïò
floor = 5;
Waitnum_0 = zeros(max(stair_num),floor);
Start_time = zeros(max(stair_num),floor);
Finish_time = zeros(max(stair_num),floor);

%% 2
% ţÃţ¡ÖÕÖźÊśijäčňţÈţÈt'ýÈËÊýÖõžó£ÉÒÔÈůűÍÊśijäžÍÕûÌåÂěÌÝźd'Œžŕ Êý
People1_num = textread('people4.txt','%d');
[data1,data2,data3,data4] = textread('costs4.txt','%f %f %f %f');
Distance1 = [data1,data2,data3,data4];
Distance1 = 40.*Distance1;
[Allocation1,Result1,Duty1,Start_time(1:4,1),Finish_time(1:4,1),Wait_num1,People_get1,Wait_num_

%% 1
People2_num = textread('people3.txt','%d');
[data1,data2,data3,data4,data5] = textread('costs3.txt','%f %f %f %f %f');
Distance2 = [data1,data2,data3,data4,data5];
Distance2 = 40.*Distance2;
[Allocation2,Result2,Duty2,Start_time(:,2),Finish_time(:,2),Wait_num2,People_get2,Wait_num_uc2]

%% 0
People3_num = textread('people2.txt','%d');
[data1,data2,data3] = textread('costs2.txt','%f %f %f');
Distance3 = [data1,data2,data3];
Distance3 = 40.*Distance3;
[Allocation3,Result3,Duty3,Start_time(1:3,3),Finish_time(1:3,3),Wait_num3,People_get3,Wait_num_

%% -1
People4_num = textread('people1.txt','%d');
[data1,data2,data3] = textread('costs1.txt','%f %f %f');
Distance4 = [data1,data2,data3];
Distance4 = 40.*Distance4;
[Allocation4,Result4,Duty4,Start_time(1:3,4),Finish_time(1:3,4),Wait_num4,People_get4,Wait_num_
```

```
%% ÕûÌåt'ęÀí
clf;
People_get1(101:300,:) = ones(200,4).*People_get1(100,:);%Àl'Õź
People_get2(101:300,:) = ones(200,5).*People_get2(100,:);
People_get3(101:300,:) = ones(200,3).*People_get3(100,:);
time_span1(:) = Finish_time(:,1) - Start_time(:,1);
time_span2(:) = Finish_time(:,2) - Start_time(:,2);%ÈůűÍÀl'ÕźÊśijä
time_span3(:) = Finish_time(:,3) - Start_time(:,3);
Down_Time = 15;%ÏÂÒżšãÂě ţÄÊśijä
Down_Speed = 5;
for stair = 1:5
    for time = 1:time_span2(stair)
        People_get2(Finish_time(stair,1)+Down_Time+time,stair) = People_get2(Finish_time(stair,
Down_Speed;
    end
    People_get2(Finish_time(stair,1)+Down_Time+time_span2(stair):300,stair) = ones(301-(Finish_
%     for time = 1:length(People_get2)-Start_time(:,2)
%         if time < time_span2(stair)
%             People_get2(Finish_time(stair,1)+Down_Time+time,stair) = People_get2(Finish_time(
Down_Speed;
%         else
%             People_get2(Finish_time(stair,1)+Down_Time+time,stair) = People_get2(Finish_time(
%         end
%     end
    figure(stair);
    plot(People_get2(:,stair));
end
Wait_num_final = 0.*People_get2;
Num_left_reg = 0;
time_temp = 0;
for stair = 1:5
    for time = 2:300
        Num_left_reg = Num_left_reg + People_get2(time,stair)-People_get2(time-1,stair)-Down_Sp
        if Num_left_reg < 0
            Num_left_reg = 0;
        end
        Wait_num_final(time,stair) = Num_left_reg;
    end
    figure(stair+5);
    plot(Wait_num_final(:,stair));
    %axis([0;100;-5;280]);
    xlabel('Loop times'), ylabel('The number of waiting people');
end
for i =1:4
    figure(10+i)
    hold on;
    plot(Wait_num1(:,i),'r');
    plot(Wait_num_uc1(:,i),'b');
    xlabel('Loop times'), ylabel('The number of waiting people');
    legend('Allocation','No Allocation');
    axis([0 150 0 550]);
    hold off
end
%% write the data
fid=fopen('Allocation1.txt','w');
[r,c]=size(Allocation1);            % ţÃţ¡¿ØÕóţÄÐÐÊýžÍÁÐÊý
 for i=1:r
  for j=1:c
  fprintf(fid,'%f\t',Allocation1(i,j));
  end
  fprintf(fid,'\r\n');
```

```
 end
fclose(fid);
clear r;
clear c;

fid=fopen('Allocation2.txt','w');
[r,c]=size(Allocation2);            % ţÃţ¡¿ØÕótÄÐÐÊýžÍÁÐÊý
 for i=1:r
  for j=1:c
  fprintf(fid,'%f\t',Allocation2(i,j));
  end
  fprintf(fid,'\r\n');
 end
fclose(fid);
clear r;
clear c;

fid=fopen('Allocation3.txt','w');
[r,c]=size(Allocation3);            % ţÃţ¡¿ØÕótÄÐÐÊýžÍÁÐÊý
 for i=1:r
  for j=1:c
  fprintf(fid,'%f\t',Allocation3(i,j));
  end
  fprintf(fid,'\r\n');
 end
fclose(fid);
clear r;
clear c;

fid=fopen('Allocation4.txt','w');
[r,c]=size(Allocation4);            % ţÃţ¡¿ØÕótÄÐÐÊýžÍÁÐÊý
 for i=1:r
  for j=1:c
  fprintf(fid,'%f\t',Allocation4(i,j));
  end
  fprintf(fid,'\r\n');
 end
fclose(fid);
clear r;
clear c;
%%
Result1 = Result1';
Result2 = Result2';
Result3 = Result3';
Result4 = Result4';
```

Appendix H The result of the allocation calculation

Table 8: The allocation result of the 2nd floor

	exit1	exit2	exit3	exit4
hall1	0.367498	0.367177	0.247988	0.017336
hall2	0.118045	0.148752	0.346315	0.386888
hall3	0.016247	0.518165	0.150473	0.315115
hall4	0.379608	0.03707	0.268768	0.314553
hall5	0.062027	0.440672	0.389008	0.108293
hall6	0.184424	0.569187	0.064306	0.182083
hall7	0.059779	0.377837	2.17E-05	0.562362
hall8	0.473997	0.048377	0.477624	2.15E-06
hall9	0.34432	0.290488	0.003229	0.361962

Table 9: The allocation result of the 1st floor

	exit1	exit2	exit3	exit4	exit5
hall1	0.515973	0.359711	0.072548	0.050976	0.000793
hall2	0.061884	0.043843	0.195845	0.447013	0.251414
hall3	0.294452	0.132199	0.180509	0.12308	0.26976
hall4	0.335466	0.338974	0.00382	0.097113	0.224627
hall5	0.229306	0.18907	0.100553	0.328905	0.152165
hall6	0.165302	0.21059	0.14668	0.263898	0.21353
hall7	0.094742	0.151152	0.455744	0.139645	0.158717
hall8	0.375494	0.295801	0.187572	1.61E-06	0.141131
hall9	0.052895	0.041466	0.223555	0.406611	0.275472
hall10	0.172995	0.279777	0.123744	0.09256	0.330924
hall11	0.080733	0.338674	0.024878	0.037344	0.518372
hall12	0.369784	0.024219	0.259034	0.226821	0.120142
hall13	0.149316	0.131986	0.230257	0.436409	0.052032
hall14	0.095861	0.001107	0.37621	0.482285	0.044537
hall15	0.040256	0.125695	0.377718	0.369412	0.08692
hall16	0.019606	0.372431	0.090148	0.181736	0.336079

Table 10: The allocation result of the ground floor

	exit1	exit2	exit3
hall1	0.95619	0.043807	2.84E-06
hall2	0.25971	0.535497	0.204793
hall3	0.155332	0.428264	0.416404
hall4	0.066531	0.699183	0.234286
hall5	0.125035	0.180744	0.694221
hall6	0.593476	0.165059	0.241465
hall7	0.151736	0.69057	0.157693
hall8	0.261916	0.167703	0.570381
hall9	0.265006	0.291244	0.44375
hall10	0.398601	0.455146	0.146252
hall11	0.565989	0.128948	0.305063
hall12	0.578715	0.068936	0.352349
hall13	0.318103	0.437515	0.244382
hall14	0.511288	0.308934	0.179778
hall15	0.102287	0.236197	0.661516

Table 11: The allocation result of the B1 floor

	exit1	exit2	exit3
hall1	0.413655	0.005772	0.580573
hall2	0.824708	0.00411	0.171182
hall3	8.15E-05	0.082739	0.917179
hall4	0.284614	0.112785	0.602601
hall5	0.499641	0.473834	0.026525
hall6	0.509328	0.20636	0.284312
hall7	0.159304	0.449778	0.390918
hall8	0.212471	0.504035	0.283494
hall9	0.709129	0.106664	0.184207
hall10	0.26948	0.55083	0.17969
hall11	0.218473	0.728556	0.052971
hall12	0.514532	0.438161	0.047308
hall13	0.006879	0.600818	0.392303
hall14	0.000518	0.99859	0.000892

Appendix I The Flow Chart of making evacuation plan

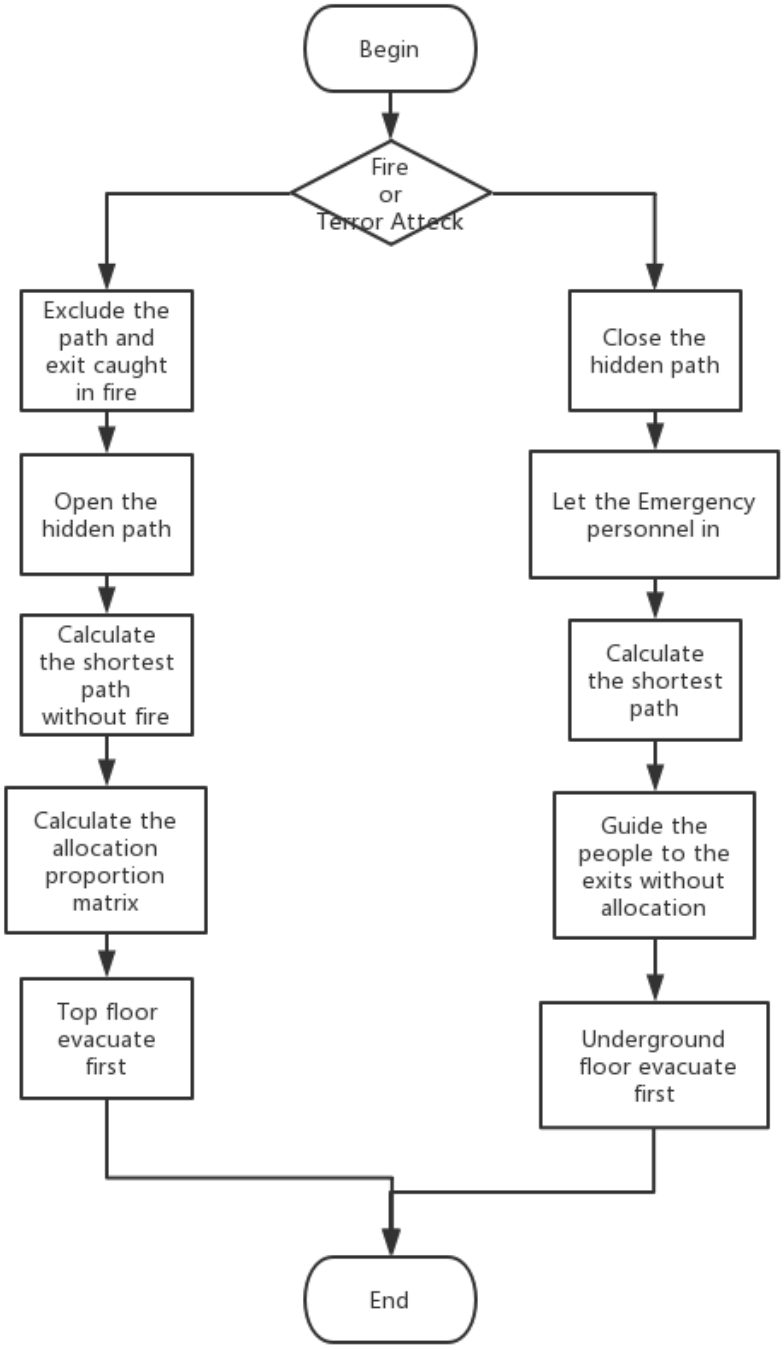

Figure 34: Overall Flow Chart of making evacuation plan